МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«ВЛИЯНИЕ КЭШ-ПАМЯТИ НА ВРЕМЯ ОБРАБОТКИ МАССИВОВ»

студента 2 курса, 23202 группы

Пятанова Михаила Юрьевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
Владислав Александрович
Перепёлкин

Новосибирск 2024

СОДЕРЖАНИЕ

ЦЕЛЬ.....3
ОПИСАНИЕ РАБОТЫ 4
ЗАКЛЮЧЕНИЕ.....5
Приложения 1, 2, 3. 6

ЦЕЛЬ

1. Исследование зависимости времени доступа к данным в памяти от их объема.

2. Исследование зависимости времени доступа к данным в памяти от порядка их обхода.

ОПИСАНИЕ РАБОТЫ

В ходе выполнения лабораторной работы, были написаны три различных метода обхода массива: прямой, обратный и случайный, реализация которых представлена в приложении 1, функции “fill”.

Отдельно стоит обратить внимание на создание массива, заполненного случайно. Так, чтобы при обходе массива участвовали все элементы, был написан специальный алгоритм, действующий следующим образом: создается новый массив, заполненный прямым образом, далее он перемешивается. Изначальный массив заполняется так: считывается элемент x в перемешанном массиве, далее считывается следующий после него (следующим элементом для последнего считается элемент с индексом 0), в начальный массив вставляется в x-ую позицию этот следующий элемент.

Код реализованной программы, измеряющей среднее время доступа к одному элементу при разных способах заполнения массива, представлен в приложении 1. Результаты измерений записываются в файлы “out0.txt”, “out1.txt” и “out2.txt”. На основе их с помощью библиотеки python “matplotlib” (приложение 2) были построены несколько графиков: без флагов оптимизации и с -O1 (приложение 3).

По полученным результатам можно сделать следующие выводы (все выводы на одно ядро процессора):

1) Размер L1 кэша данных – 32 КБ (по совместительству размер массива, после которого время доступа к элементу при случайном обходе больше, чем при прямом и обратном, ведь он не помещается полностью в L1 кэш и контроллер не может предсказать какой следующий блок загружать).

2) Размер L2 кэша – 512 КБ

3) Размер L3 кэша - 3,215 МБ

Что при сравнении с известными реальными значениями практически совпадает (приложение 3).

ЗАКЛЮЧЕНИЕ

Исследованы зависимости времени доступа к данным в памяти от их объема, исследованы зависимости времени доступа к данным в памяти от порядка их обхода.

На основе полученных данных были сделаны выводы о размере кэша разных уровней:

1) Размер L1 кэша данных – 32 КБ
2) Размер L2 кэша – 512 КБ
3) Размер L3 кэша - 3,215 МБ

Что при сравнении с известными реальными значениями практически совпадает – различается лишь кэш L3, что связано с его unified статусом. Контроллер просто не дает его полностью заполнить, оставляя место для других ядер.

Приложение 1.

Команда для компиляции:
g++ lab8.c -o ex
g++ lab8.c -o ex -O1

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <random>
#include <chrono>

enum class Mode {forward = 0, backward, random};

// 1 KB.
const int minN = 256;
// 32 MB.
const int maxN = 8388608;

void fill(int* array, size_t size, Mode mode) {
    if (mode == Mode::forward) {
        for (size_t i = 0; i < size - 1; ++i) {
            array[i] = i + 1;
        }
        array[size - 1] = 0;
        return;
    }
    if (mode == Mode::backward) {
        for (size_t i = size - 1; i > 0; --i) {
            array[i] = i - 1;
        }
        array[0] = size - 1;
        return;
    }
    int* temp = new int[size];
    for (size_t i = 0; i < size - 1; ++i) {
        temp[i] = i + 1;
    }
    temp[size - 1] = 0;
    std::mt19937 g(time(nullptr));
    std::shuffle(temp, temp + size, g);
    for (size_t i = 0; i < size - 1; i++) {
        array[temp[i]] = temp[i + 1];
    }
    array[temp[size - 1]] = temp[0];
```

```
    delete[](temp);
}

void progrev(int* array, size_t size) {
    for (size_t k = 0, i = 0; i < size; i++) {
        k = array[k];
    }
}

uint64_t getCpuTicks() {
    unsigned int lo, hi;
    __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
    return ((unsigned long long)hi << 32) | lo;
}

uint64_t measure(int* array, size_t size) {
    size_t k, i;
    uint64_t start = getCpuTicks();
    for (k = 0, i = 0; i < size; i++) {
        k = array[k];
    }
    uint64_t end = getCpuTicks();
    return (end - start) / size;
}

int main() {
    const size_t iterations = 7;
    std::ofstream out0;
    out0.open("out0.txt");
    std::ofstream out1;
    out1.open("out1.txt");
    std::ofstream out2;
    out2.open("out2.txt");
    int* a = new int[maxN];
    size_t inc = 1;
    size_t border = 1;
    for (size_t n = minN; n < maxN; n += inc) {
        for (Mode mode : {Mode::forward, Mode::backward,
Mode::random}) {
            fill(a, n, mode);
            progrev(a, n);
            uint64_t minTicks = UINT64_MAX;
            for (size_t i = 0; i < iterations; ++i) {
                uint64_t ticks = measure(a, n);
```

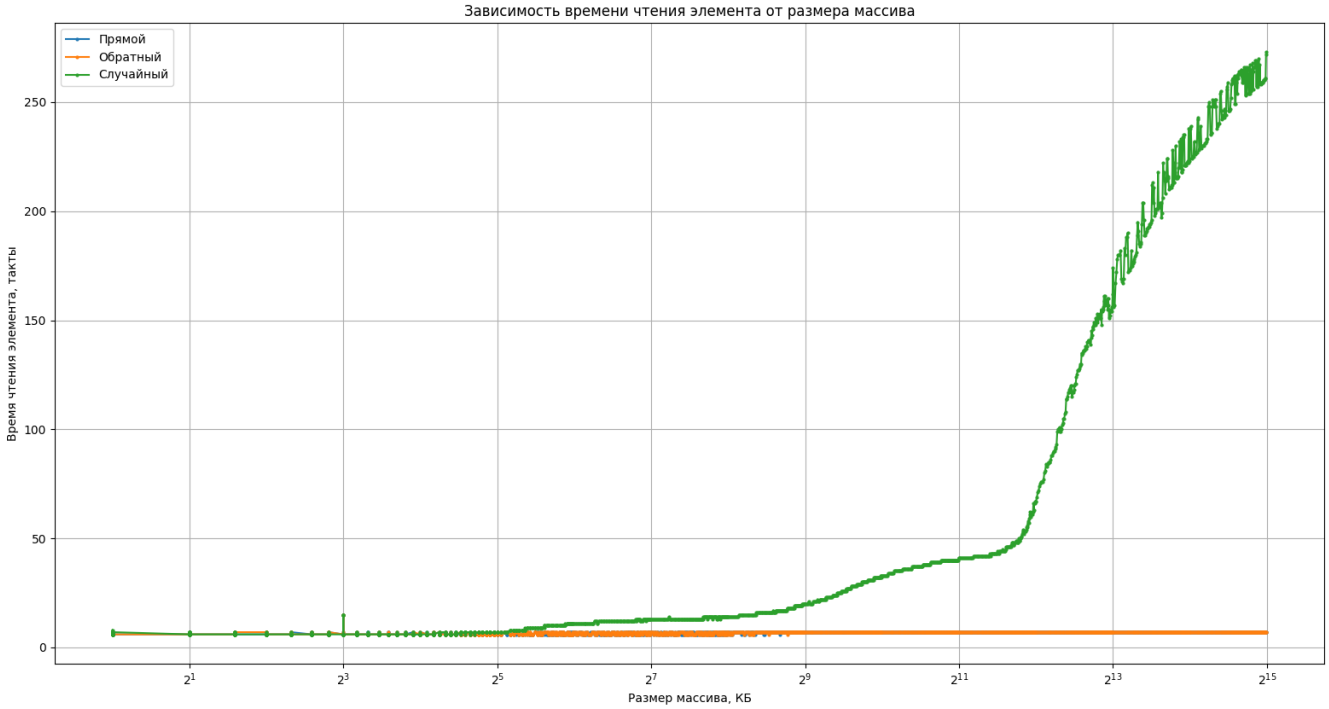
```
                minTicks = minTicks > ticks ? ticks : minTicks;
            }
            switch (mode) {
            case Mode::forward:
                out0 << (n * 4) / 1024 << ' ' << minTicks << '\n';
                break;
            case Mode::backward:
                out1 << (n * 4) / 1024 << ' ' << minTicks << '\n';
                break;
            case Mode::random:
                out2 << (n * 4) / 1024 << ' ' << minTicks << '\n';
                break;
            default:
                break;
            }
        }
        if ((n * 4) / 1024 >= border){
            inc <<= 1;
            border <<= 1;
        }
    }
    delete[](a);
    out0.close();
    out1.close();
    out2.close();
    return 0;
}
```

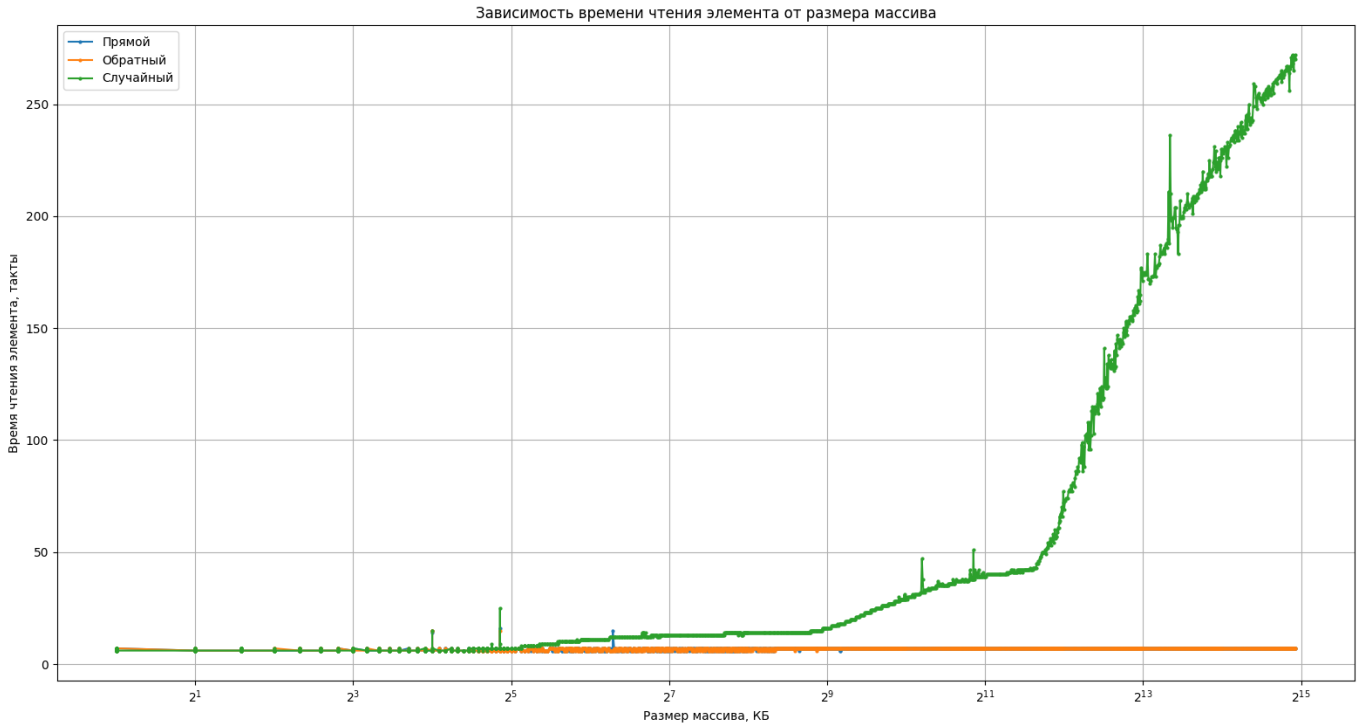
Приложение 2.

```
import matplotlib.pyplot as plt
import numpy as np

def read_data(file_path):
    x_values = []
    y_values = []
    with open(file_path, 'r') as file:
        for line in file:
            x, y = map(float, line.strip().split())
            x_values.append(x)
            y_values.append(y)
    return x_values, y_values

def plot_data(data_list, labels):
    for (x_values, y_values), label in zip(data_list, labels):
        plt.plot(x_values, y_values, marker='o', linestyle='-',
label=label, markersize=2)
    plt.xscale('log', base = 2)
    plt.xlabel('Размер массива, КБ')
    plt.ylabel('Время чтения элемента, такты')
    plt.title('Зависимость времени чтения элемента от размера
массива')
    plt.grid(True)
    plt.legend()
    plt.show()

if __name__ == "__main__":
    file_paths = ['out0.txt', 'out1.txt', 'out2.txt']
    labels = ['Прямой', 'Обратный', 'Случайный']
    data_list = [read_data(file_path) for file_path in file_paths]
    plot_data(data_list, labels)
```

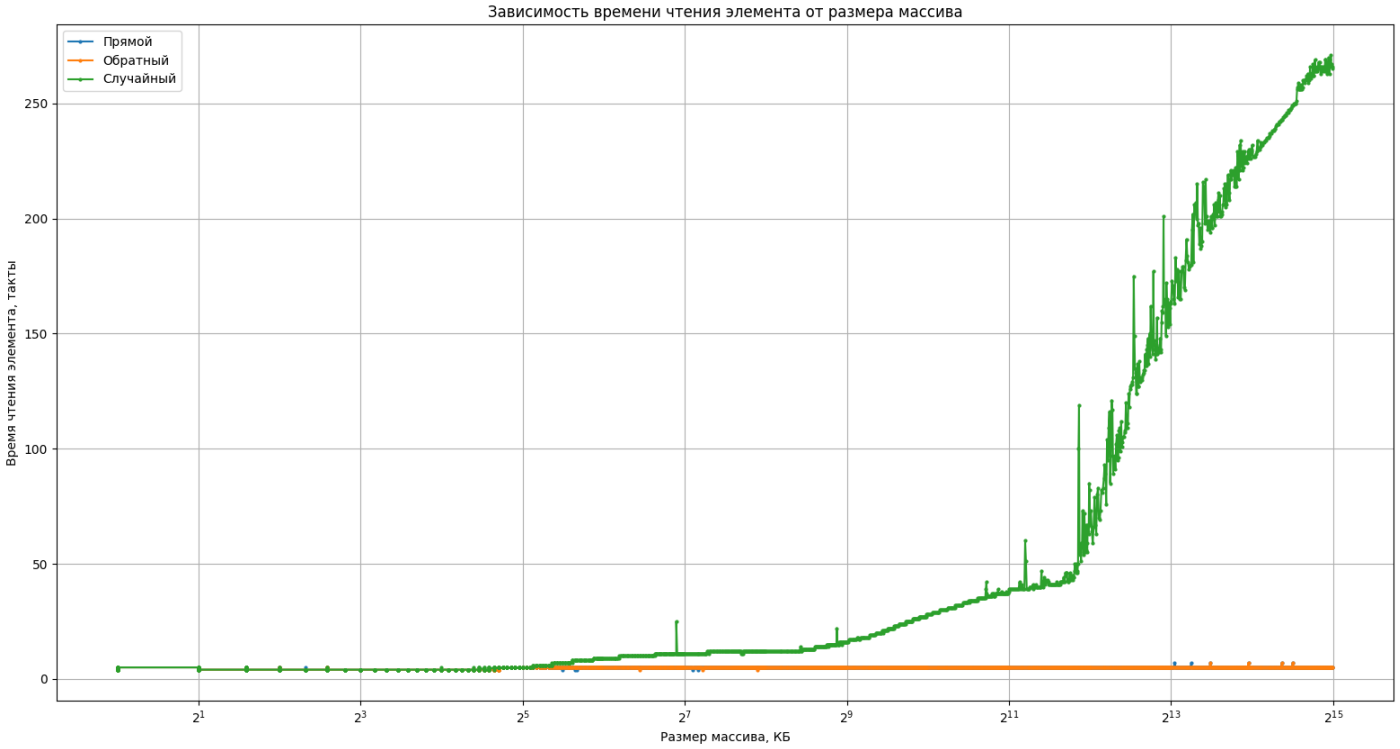
Приложение 3.

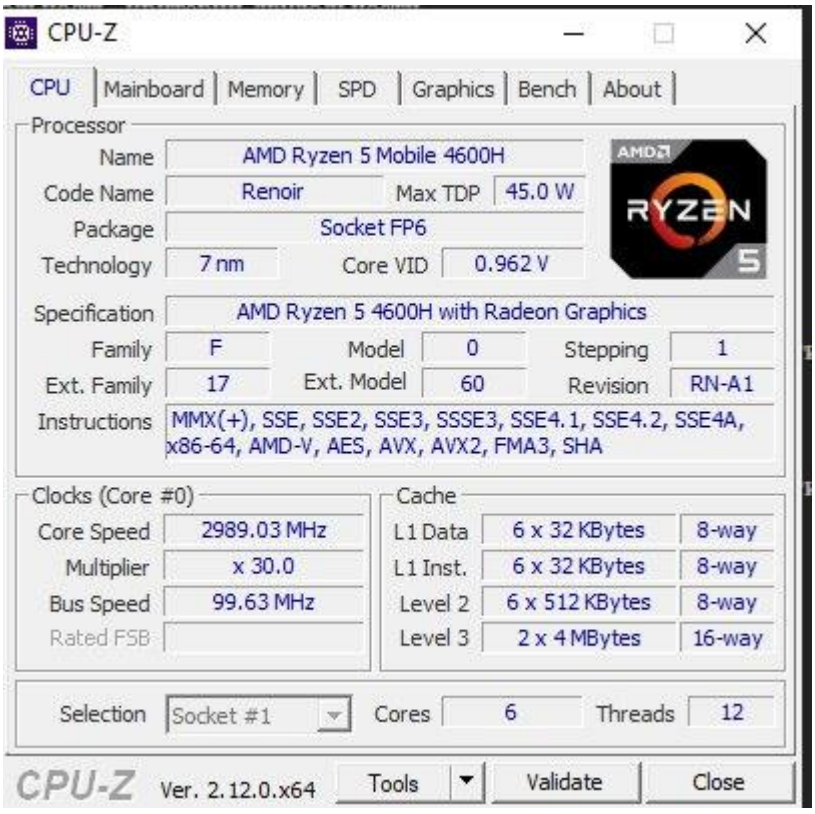
Без флага оптимизации -O1:





С флагом:


Зависимость времени чтения элемента от размера массива
Прямой
Обратный
Случайный
Время чтения элемента, такты
Размер массива, КБ
0
50
100
150
200
250
2^1
2^3
2^5
2^7
2^9
2^{11}
2^{13}
2^{15}


CPU-Z
CPU
Mainboard
Memory
SPD
Graphics
Bench
About
Processor
Name
AMD Ryzen 5 Mobile 4600H
Code Name
Renoir
Max TDP
45.0 W
Package
Socket FP6
Technology
7 nm
Core VID
0.962 V
Specification
AMD Ryzen 5 4600H with Radeon Graphics
Family
F
Model
0
Stepping
1
Ext. Family
17
Ext. Model
60
Revision
RN-A1
Instructions
MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A, x86-64, AMD-V, AES, AVX, AVX2, FMA3, SHA
Clocks (Core #0)
Core Speed
2989.03 MHz
Multiplier
x 30.0
Bus Speed
99.63 MHz
Rated FSB
Cache
L1 Data
6 x 32 KBytes
8-way
L1 Inst.
6 x 32 KBytes
8-way
Level 2
6 x 512 KBytes
8-way
Level 3
2 x 4 MBytes
16-way
Selection
Socket #1
Cores
6
Threads
12
CPU-Z
Ver. 2.12.0.x64
Tools
Validate
Close